WeatherNet

Developer Manual

Dor Shabat and Yuval Rozner

BRAUDE College of Engineering, Karmiel

February 04, 2025

BRAUDE
College of Engineering, karmiel

Contents

Introduction

Purpose of the Developer Manual

The WeatherNet Developer Manual is designed to provide developers with comprehensive guidance on understanding, modifying, and extending the WeatherNet platform. Whether you are contributing to the backend machine learning model, enhancing the user interface, or deploying the application, this manual offers detailed instructions and best practices to ensure efficient and effective development.

Overview of WeatherNet Architecture

WeatherNet is a powerful weather forecasting platform that combines a machine learning-based backend with an intuitive React web application frontend. Its architecture is designed for accuracy, scalability, and ease of use, ensuring that users have access to real-time, reliable weather predictions.

Backend: Machine Learning

The backend of WeatherNet leverages advanced machine learning techniques to deliver accurate mid-term weather forecasts. Built using PyTorch, the backend architecture integrates Convolutional Neural Networks (CNNs) and Transformers Encoding with few Fully Connected Layers models to process and predict weather data effectively. Key features include:

- **Data Handling:** Supports preprocessing and management of meteorological datasets.
- **Model Training and Evaluation:** Implements robust pipelines for training and testing predictive models.
- **Deployment:** Provides integration with Firebase and other backend services for seamless interaction with the frontend.

Client Side: React Web App

The frontend is a React-based web application that offers a user-friendly interface for accessing WeatherNet's features. Key highlights include:

- **Dynamic UI:** Responsive design with light/dark theme options.
- **Interactive Features:** Real-time forecasts, comparison tools, and statistical insights.
- **Efficient Codebase:** Utilizes modern development tools like styled-components and Material-UI for streamlined styling and functionality.

Backend: Machine Learning Model

Installation and Setup

Required Python version and dependencies

The WeatherNet backend is built using Python and relies on several key libraries for machine learning, data processing, and evaluation. Ensure you have **Python 3.8** or later installed on your system. Below are the main dependencies used in the project:

- **PyTorch**: For building and training the model.
- **NumPy**: For numerical operations and data manipulation.
- **pandas**: For data loading and preprocessing.
- **scikit-learn**: For data scaling and evaluation metrics.
- **tqdm**: For progress bars during training and inference.
- **matplotlib**: For visualizing predictions and results.

Dependencies can be found in the requirements.txt file and installed with a single command.

Installation guide

To set up the backend environment, follow these steps:

1. **Clone the Repository**:
```
git clone <repository_url>
cd WeatherNet/backend/Model_Pytorch
```
2. **Create and Activate a Virtual Environment**:
```
python -m venv env
.\env\Scripts\activate
```
3. **Install Dependencies**: Use the requirements.txt file to install all necessary libraries:
```
pip install -r requirements.txt
```
4. **Verify Installation**: Run the following command to check that all dependencies are installed correctly:
```
python -c "import torch; print(torch.__version__)"
```
This should print the version of PyTorch installed.

Data

About the Dataset

The WeatherNet backend processes historical weather data to train its machine learning model. The dataset contains measurements from weather stations taken from the 'Israel Meteorological Service', including temperature, wind speed, wind direction, and additional time-based features. Data is provided in pickle format and is structured with columns representing different meteorological variables and a timestamp column.

Key dataset features include:

- **Year**: Year of the recorded measurement.
- **Year sin/Year cos**: Cyclic sin and cos transformations for the year.
- **Day sin/Day cos**: Cyclic sin and cos transformations for the day of the year.
- **TD (degC)**: Dew point temperature in degrees Celsius.
- **TDmax/TDmin (degC)**: Maximum and minimum dew point temperatures.
- **Rain**: amount of rain in mm.
- **RH (%)**: Relative humidity percentage.
- **Wind_x/Wind_y**: Vercoric representation of wind velocity and direction.
- **Gust_x/Gust_y**: Vercoric representation of wind gust velocity and direction.
- **STDwd (deg)**: Standard deviation of wind direction.

Preprocessing

Data preprocessing is handled using utilities in data.py and follows these steps:

1. **Handling Missing Data**:
   - Missing values are handled using interpolation.
2. **Feature Engineering**:
   - Wind speed and direction are transformed into vector components (Wind_x, Wind_y) for better modeling.
   - Wind gusts are similarly decomposed into Gust_x and Gust_y.
   - Time-based cyclic features (Day sin, Day cos, Year sin, Year cos) are created to encode temporal patterns effectively.
3. **Normalization**:
   - All numeric fields are normalized using StandardScaler from scikit-learn to ensure features are on a comparable scale, improving model performance.

Architecture

Architecture Overview

The WeatherNet Model is designed to predict weather parameters for a specific station by effectively capturing spatial and temporal dependencies from nearby stations.

The architecture integrates ***1D Convolutional Neural Networks (CNNs)*** for feature extraction, positional encodings for spatial and temporal context, and a ***Transformer Encoder*** to model complex interactions between stations and across time windows. The final prediction layer outputs the desired weather metrics.

Hybrid Model Approach

We employ a **multi-model strategy**, utilizing four instances of the **WeatherNet Model**, each optimized for different forecasting horizons. While all models share the same architecture, they specialize in distinct time frames to enhance predictive accuracy:

- **0–12 hours**: Captures short-term weather dynamics.
- **12–24 hours**: Provides insights into near-daily trends.
- **24–36 hours**: Focuses on mid-term patterns.
- **36–60 hours**: Optimized for longer-range forecasts.

By training each model on its respective time window, we ensure that learned representations are tailored to specific temporal dependencies, improving accuracy across different forecasting horizons.

Training

The training process for the WeatherNet model is defined in train.py.

Key steps include:

1. **Data Preparation**:
   - Sliding windows of historical data are generated using window_generator.py.
   - These windows are fed into the model to predict the next set of time steps.
2. **Loss Function**:
   - The model minimizes the Mean Squared Error (MSE) loss function, which measures the accuracy of predictions compared to actual values.
3. **Optimization**:
   - Uses the Adam optimizer for efficient training with adaptive learning rates.
4. **Checkpoints**:
   - Model checkpoints are saved during training to ensure progress is not lost.

- The best-performing model is saved as best_checkpoint.pth in the output/checkpoints/ directory.

5. **Training Loop**:
   - Iterates over epochs, with progress tracked using tools like tqdm for logging.

Inference

Inference, or making predictions with the trained model, is handled in inference.py.

Key steps include:

1. **Loading the Trained Model**:
   - The best model checkpoint is loaded from output/checkpoints/best_checkpoint.pth.
2. **Generating Predictions**:
   - The input data is processed into sliding windows, consistent with the training setup.
   - Predictions are made for each window and aggregated into a continuous forecast.
3. **Post-Processing**:
   - Predictions are converted back to the original scale using the scaler saved during preprocessing.
   - Results can be visualized or exported in formats such as CSV for integration with the frontend.

By combining an effective architecture with robust training and inference workflows, the WeatherNet backend achieves accurate and reliable weather forecasts.

Evaluation & Validation

Evaluation is a critical step in ensuring the accuracy and reliability of the WeatherNet machine learning model. The backend includes well-defined metrics and validation strategies to measure the model's performance effectively.

Evaluation and metrics

The evaluation process is centered around comparing the model's predictions against actual observed data. The following metrics and methods are used:

1. **Mean Squared Error (MSE)**:
   - This is the primary evaluation metric.
   - It measures the average squared difference between the predicted and actual values:

```
MSE = (1/n) * Σ (y_true - y_pred)^2
```

Lower values indicate better model performance.

2. **Root Mean Squared Error (RMSE)**:
   - RMSE is derived from MSE and provides an interpretable metric in the same scale as the target variable.

```
RMSE = √MSE
```

3. **Validation Loss**:
   - During training, the model evaluates its performance on a validation set after each epoch.
   - Validation loss (calculated using MSE) is used to monitor overfitting and determine the stopping point for training.

Validation Process

1. **Train-Test Split**:
   - The dataset is divided into training, validation, and test sets.
2. **Cross-Validation**:
   - Cross-validation is occasionally applied to ensure robust evaluation by training and testing on different data subsets.
3. **Visualization:**
   - Results are visualized to provide insights into model performance. Graphs such as predicted vs. actual values and error distributions help in understanding the accuracy of predictions.

Improving Model and Training

The WeatherNet backend is designed with flexibility, allowing developers to enhance the model and optimize training processes. The key aspects of improving the model and training include:

Modifying the Model Architecture

The architecture of the machine learning model is defined in the model.py file.

Developers can:

- Modify existing components, such as the neural-network layers, dense layers, or activation functions.
- Add new layers or features to better capture weather data patterns.
- Experiment with different architectures to test performance improvements.

Changes made in model.py are automatically reflected during the next training cycle, enabling quick experimentation and iteration.

Hyperparameter Tuning

Hyperparameters play a crucial role in the model's performance and training efficiency. Key hyperparameters, defined in the parameters.py file, include:

- **Learning Rate**: Controls the step size during optimization.
- **Batch Size**: Determines the number of samples processed per training step.
- **Number of Epochs**: Sets the total number of iterations over the dataset.
- **Window Size**: Defines the length of the input sequence time.

To tune hyperparameters:

1. Update values in the parameters.py file.
2. Rerun the training script (train.py) to evaluate the impact of changes.
3. Use validation metrics like MSE to assess performance improvements.

By leveraging these modification and tuning capabilities, developers can iteratively refine the model to achieve higher accuracy and adapt it to new requirements or datasets.

Client Side: React Web App

Setting Up the Development Environment

Required tools

To begin developing or maintaining the WeatherNet React Web App, you need the following tools installed on your system:

- **Node.js and npm:** Ensure that you have Node.js (version 16 or later) and npm installed. These are essential for managing dependencies and running the development server.
- **Code Editor:** Visual Studio Code (VS Code) is recommended for its support of JavaScript/React and its rich extensions.
- **Git:** Version control is managed using Git, and a Git client or terminal is necessary to clone repositories and manage branches.

Project Dependencies and Modules

The WeatherNet React Web App leverages several modern libraries and tools to streamline development and enhance functionality. Below are the key dependencies and their roles:

Core Dependencies

- **React (v19.0.0):** The core library for building the application's UI.
- **React-DOM (v19.0.0):** Enables React to render the application in the browser.

Styling

- **@emotion/react and @emotion/styled:** Provides a flexible styling solution.
- **Styled-components (v6.1.14):** A modern way to handle component-level styling.
- **@mui/material and @mui/icons-material:** Material-UI libraries for pre-built, customizable UI components and icons.

Functionality Enhancements

- **Firebase (v11.2.0):** Used for deployment and backend services.
- **React-router-dom (v7.1.2):** Manages client-side routing.
- **React-share (v5.1.2):** Enables easy sharing of content across platforms.
- **React-icons (v5.4.0):** Offers a variety of icons for improved UI design.
- **React-draggable (v4.4.6) and React-resizable (v3.0.5):** Provides drag-and-drop and resizing functionalities.

Running the development server

To run the WeatherNet application locally:

1. Clone the WeatherNet repository using Git.

   ```
   git clone <https://github.com/YuvalRozner/WeatherNet>
   ```

2. Navigate to the project directory:

   ```
   cd ./WeatherNet Website/weathernet/
   ```

3. Install all required dependencies:

   ```
   npm install
   ```

4. Start the development server:

   ```
   npm start
   ```

   This command will launch the application on http://localhost:3000/ by default. The development server provides hot-reloading, allowing you to see changes in real-time.

Building for production

To build the application for production:

1. Run the build script:

   ```
   npm run build
   ```

   This will create an optimized production-ready build in the build/ directory.

2. Deploy the build to Firebase or your preferred hosting platform using the deployment command.

Project Structure

The WeatherNet React Web App is organized into two primary directories: src and public. Below is an overview of their structure and purpose.

Overview of the src folder

The src folder contains the core application logic and components. It is structured to promote modularity and maintainability.

Components

The components directory includes reusable and page-specific components, subdivided into:

- **Base Components:** Found in components/base, these provide foundational UI elements such as:
    - baseLayout.js: Manages the general layout structure.
    - weathenetLogoLayout.js: Displays the WeatherNet logo.
    - helpDialog.js: Provides user assistance pop-ups.
- **Pages:** Located in components/pages, these are the main pages of the application, including:
    - about/: Includes about.js and styling files for the "About" page.
    - architecture/: Includes architecture.js for showcasing the platform architecture.
    - contactUs/: Handles the contact form in contactUs.js.
    - statistics/: Displays forecast statistics using statistics.js and supplementary components like tables.js.
    - weatherForecast/: Includes weathenetForecast.js and imsForecast.js for displaying forecasts.
- **Top Bar Content:** Contains files like topBar.js and githubContainer.js to manage the header section.

Utils

The utils directory houses helper modules and configurations:

- theme.js: Manages application themes (light/dark).
- logger.js: Provides logging utilities for debugging.
- network/: Contains files like weathernetConnection.js and ImsConnection.js to handle API requests.

Overview of the public folder

The public folder holds static assets and configurations required for the application to run efficiently.

HTML templates

index.html: The main HTML file that serves as the entry point for the React app. It includes links to scripts and metadata.

Assets

- **Images and Logos:** Found in subfolders like logo/ and figures/. Examples include:
    - compressed_logo.png: Used in the header.
- **Documentation:** The papers/ folder contains PDF files such as:
    - WeatherNet - User Manual.pdf
    - WeatherNet - Developer Manual.pdf

Deployment

Deploying the WeatherNet React Web App is a straightforward process that uses Firebase Hosting. Below are the steps for setting up and managing the deployment.

Deployment to Firebase

Prerequisites

Before deploying, ensure the following:

- Firebase CLI is installed. If not, install it using:

```
npm install -g firebase-tools
```

- Your Firebase account is linked to your CLI by running:

```
firebase login
```

Deployment Steps

Log In: Ensure you are logged into Firebase via the CLI:

```
firebase login
```

Deploy: Deploy the latest version of the application to Firebase Hosting:

```
firebase deploy
```

After deployment, Firebase provides a unique URL for your app.

Managing updates and version control

Updating the App

To update the deployed app:

1. Make changes to the codebase.
2. Rebuild the application:

```
npm run build
```

3. Deploy the updated build:

```
firebase deploy
```

Version Control

- Use Git for managing source code. Commit changes frequently with clear commit messages.
- Use branches for new features or fixes, and merge them into the main branch after review.

Key Components and Their Functionality

The WeatherNet React Web App is built with several key components that form the foundation of its functionality. Below is an overview of these components and their roles within the application.

Base Components

baseLayout.js

Manages the overall layout and structure of the application, including the placement of navigation bars, content areas, and footers.

weathenetLogoLayout.js

Handles the display of the WeatherNet logo, ensuring consistent branding across the application.

helpDialog.js

Implements pop-up dialogs that provide contextual help and guidance to users.

skeltonLayout.js

Provides skeleton loading components to enhance user experience during data fetching.

Pages

The pages directory contains the main screens of the application, each implemented with its corresponding logic and styling:

About Page (about.js)

Presents an overview of WeatherNet, its mission, and its contributors.

Architecture Page (architecture.js)

Visualizes the platform's technical architecture and provides explanatory content about the machine learning backend.

Contact Us Page (contactUs.js)

Features a contact form where users can submit inquiries or feedback. This includes fields for email, subject, and message.

Statistics Page (statistics.js)

Displays detailed statistics on forecast accuracy and performance, including interactive charts and tables for better analysis.

Weather Forecast Pages

- **WeatherNet Forecast** (weathenetForecast.js): Displays predictions generated by the WeatherNet machine learning model.
- **IMS Forecast** (imsForecast.js): Shows data from the Israel Meteorological Service for comparison.

Utilities (Utils)

The utils directory includes helper modules and utility functions to streamline development:

theme.js

Manages light and dark themes, allowing users to toggle between modes for a customized interface.

logger.js

Provides a centralized logging mechanism for debugging and monitoring application behavior only in developing mode and not in production.

network/

- weathernetConnection.js: Handles API requests and responses from the WeatherNet backend.
- ImsConnection.js: Manages data retrieval from the Israel Meteorological Service.

Adding New Features

Expanding the WeatherNet React Web App with new features is a systematic process that ensures consistency and maintainability. Below are guidelines for adding new components, styling them effectively, and updating navigation as needed.

Guidelines for creating new components

Determine whether the feature is a reusable **Base Component** (e.g., buttons, modals) or a full **Page Component** (e.g., dashboards). Then:

1. Navigate to the appropriate folder (src/components/base or src/components/pages).
2. Create a .js file for the logic and an optional .style.js file for styling.
3. Write the component as a React functional component and integrate it into the parent component or page.

- Use clear, modular naming.
- Keep components focused on a single responsibility.

Styling with style.js

The WeatherNet app uses styled-components for dynamic and maintainable styling. Follow these steps to style your new components:

1. Create a Styled File:
    - In the same directory as your component, create a file named componentName.style.js.
2. **Define Styled Components:**

```
import styled from 'styled-components';
```

3. **Apply Styles in the Component:** Import the styled components into your .js file and use them as React components:

```
import {component} from './newFeature.style';
```

4. **Use Theme Variables:** Leverage theme.js to maintain consistent colors, fonts, and spacings.

Updating navigation

To add a new feature to the navigation:

1. **Update the Navigation List:**
    - Open src/utils/navigationList.js.

      Add a new entry to the navigation array:

```
{
 segment: 'NewFeature'
 title: 'New Feature',
 icon: <NewIcon />,
 pageComponent: <NewFeature />,
},
```

2. **Automatic Route:**

   note that in the file src/App.js there is automatic implementation of Routes for new pages by the navigationList.js.
3. **Test Navigation:**

   Run the application locally to ensure the new navigation item appears in the sidebar or header and links to the correct page.

Collaborative Development

Git Workflow

The WeatherNet project is managed through a Git repository. To extend or modify the project, developers must create a new branch for their changes and submit a pull request for review. This ensures a controlled and collaborative workflow, maintaining code quality and consistency across the project.

Git Source Code Structure

All source code for both the backend (machine learning models) and the frontend (React web app) is housed in the repository. Additionally, the repository contains project documentation, including design papers and manuals, as well as supplementary materials such as diagrams and figures. This centralization of resources ensures easy access and efficient project management.

Closing Statement

Thank you for reviewing the WeatherNet Developer Manual. This guide provides key insights into the project's architecture and workflows to ensure quality and alignment with project goals.

WeatherNet thrives on teamwork and innovation. Whether enhancing backend models or improving the frontend, your contributions are essential to the platform's success.

For further assistance, consult the repository documentation or contact the team. Together, we can continue to enhance WeatherNet for accurate and impactful weather forecasting.